Aqua Research
Senior Design Project

Controller for K1 Batch Generator

Project Manager: Diego Chavez & David Kirby
with John Quinlan

The University of New Mexico
School of Engineering

Aqua Research, LLC
5601 Midway Park Place NE
Albuquerque, NM 87109

Spring 2020

Contents

List of Figures

2 Overview

2.1 Executive Summary

We set out to produce a prototype for a user interface controller to be used on the K1 Batch Generator, a product currently under development at Aqua Research, LLC. Our team faced copious issues throughout the course of development, however we were able to accurately and successfully meet each requirement and have produced a working prototype for the company. Five units have already been ordered and will be produced by July 2020.

The K1 Batch Generator is a very important product under development, and with it underdeveloped countries around the world will be able to access disinfected water through the use of chlorine produced from sodium chloride and electricity. The design of our controller was paramount to the effectiveness of the final product as without reliable hardware and software in our K1 controller, the unit would not be able to automatically monitor production.

Future work will consist of improving upon the efficiency of the code to meet even stricter guidelines. Faulty wiring in the prototyping board also caused one of the primary demuxing logic chips to not have power. Addressing this issue will allow future efforts to connect all of the designed components together for phase two prototyping.

2.2 Abstract

Aqua Research, LLC has developed systems that use electrolysis to create chlorine from basic table salt. This chlorine can then be used as a disinfectant to create potable water. With applications ranging from emergency response situations, third-world countries, and the military, it was imperative that end-users be able to remotely monitor and maintain these systems during less than ideal conditions. As most of the practical uses for the disinfectant systems are in difficult-to-access areas, it was also beneficial for Aqua Research to be able to troubleshoot these control systems while off-site.

For example, the team at Aqua Research have done tremendous work in Haiti following devastating earthquakes in 2010 and 2018. Being able to analyze these water treatment systems in Haiti remotely would greatly increase Aqua Research's presence there and its ability to help more people.

The senior design group of Aqua Research set out to create a sensor system using cost-effective components that could read data from the disinfectant tanks, detect faults, and report these findings.

The device needed to be multilingual as it will be used in a variety of countries and situations; it needed to be able to recover from power loss without losing state; and it needed to be able to communicate via long-range telecommunications.

3 Problem Description

Aqua Research, LLC develops innovative water treatment technologies that meet the extreme needs within developing countries and provides sustainable water purification to outdoor enthusiasts, travelers, emergency preppers, first responders, Peace Corps, and the military. Their expertise primarily resides in electrolytic technologies that produce disinfectants from salt to a variety of water filtration devices.

The use cases for these water treatment technologies call for the system to be remotely monitored. Our goal in this project was to design a communication system that could relay alerts via cellular GSM (global system for mobile communications) and to be able to display alert codes locally on an attached LCD and in a variety of languages. The device must be able to:

1. monitor digital and analog I/O of the disinfectant controller, including switch levels, indicator LEDs, and current/voltage levels,
2. have a simple and universal user interface and adaptable to a variety of languages,
3. recover from power loss without losing state,
4. communicate via GSM telecommunications, and
5. be cost-efficient as these devices are primarily meant for low-income and disaster areas.

A 2015 Senior Design group had previously attempted to create the K1 controller; however, they were only successful in designing a test bench, no working prototype. We took this as a challenge and set out formulating a strategy.

4 Progress Toward a Solution

The Aqua Research Senior Design Group went through many iterations of control boards. After analyzing a past group's prototype using a Beagle-Bone Black (Figure 1, page 4), we determined that we wanted to build a prototype with a newer microcontroller. We looked for one with smaller footprint and lower power consumption. Initially considering

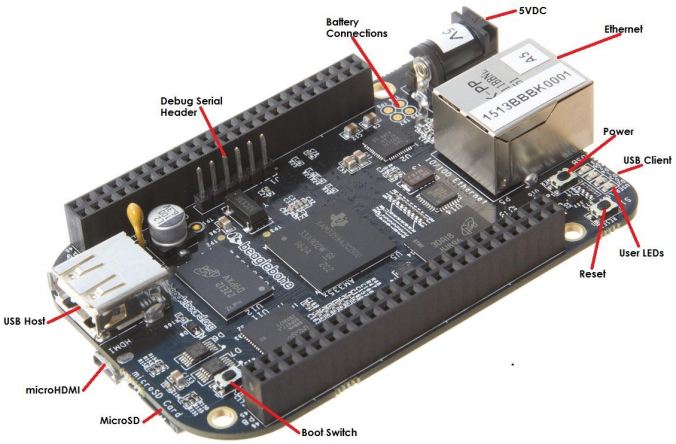

Figure 1: Beagle-Bone Black.

a Raspberry Pi configuration, the Raspberry Pi Zero W met our requirements for lower profile, built-in wireless, and was inexpensive. The Raspberry Pi 4 was powerful and had Wi-Fi support; however, neither of the Raspberry Pis natively supported analog inputs.

After discussing this with our Technical Mentor[1], it was suggested that we use the Arduino Nano as it was already implemented in other Aqua Research projects. While the last-minute change was a challenge, it proved to be the best option as it was not only less expensive but also fulfilled every requirement.

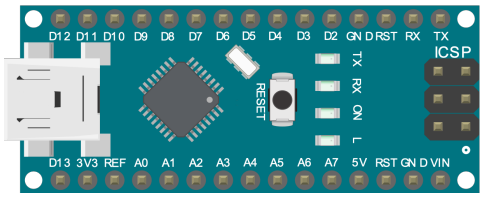

Figure 2: Arduino Nano.

After spending the first semester understanding and setting up the sensor board used to monitor the disinfectant tank, we began the second semester creating pinout mappings for the microcontroller. Using the Arduino IDE, we broke our project down into three main modules: memory, GSM, and display.

We were successful in configuring two non-volatile memory units to be used as language databases for our sensor display. Each ferroelectric RAM, or FRAM, can store up to 32 kilobytes of information. FRAMs are comparable to dynamic random-access memory but with a ferroelectric layer instead of dielectric layer. This makes it non-volatile but with the responsiveness of dynamic RAM. The FRAMs use I2C interface and one of the challenges of this setup was in connecting the components together so that we could read

[1]Tim Cushman, Aqua Research, LLC

and write to the modules individually. We accomplished changing FRAM addresses by driving specific individual pins high.

Next, we focused on the communication network. We opted for using a GSM (global system for mobile communication) breakout board, and though there were issues with the pins used, we were adamant to solve it. There were issues on which pins to use and how to address the board. The module is a Quad-band GSM/GPRS solution and can transmit SMS and data with low power consumption. Once we were successful in wiring it to the Arduino, we activated a SIM card and programmed the board to transmit using case statements based on our test scenarios.

Finally, our last requirement was to display the sensor feedback using a 4×20 LCD screen. This was also connected via I2C and allowed us to display user instructions and alerts. Together, these three modules allowed us to successfully complete our project, providing Aqua Research with a prototype which is now being implemented.

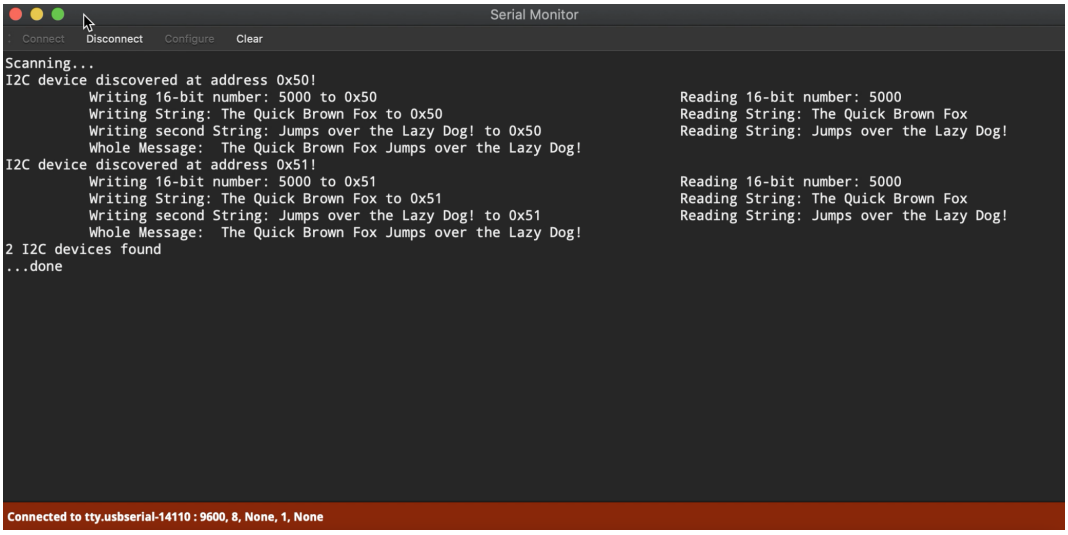

Figure 3: I2C Demonstration.

5 Constraints

The Arduino Nano (Figure 2, page 4) uses the ATmega328 chipset and has 32 KB of memory with 2 KB used for the bootloader. The ATmega328 has 2 KB of SRAM and 1 KB of EEPROM. Each of the 14 digital pins on the Nano can be used as an input or output and operate at 5 volts. The Nano has 8 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). Finally, the Nano has I2C which we used extensively for this project.

6 Budget

7 Work Schedule

8 Personnel Interactions

8.1 Teamwork

8.2 Mentorship

9 Summary & Conclusions

Spring 2020 was quite a semester for everyone across the globe and our Aqua Research Senior Design Group was no exception. The University of New Mexico shutting down all in-person operations, and the state and federal governments implementing social-distancing protocols meant working together on the project was quite a challenge.

While it is undeniable that the coronavirus had an impact on our Senior Design project, we attempted to stay focused and driven in our pursuit of creating a communication system for Aqua Research's water disinfectant tanks. We stayed in constant contact via text, email, teleconferencing software, and UNM's LoboGit repository to collaborate on the code for the project. This allowed us to make considerable progress given the circumstances, all while adhering to the stay-at-home orders.

10 Discussion

Future opportunities for this project could include programming different databases to the FRAM modules so that they may be hot-swapped as needed.

We would also like to see the code optimized as even with the extended modules we were still running low on programmable memory.

Finally, we would like to extend the code and board to be able to accept various telecommunication modules. Not everywhere uses the same network and bands, and it would be ideal to support as many areas as possible. Perhaps the GSM module could be swapped out depending upon the location where the device will be used.

11 Acknowledgements

We are grateful for our sponsors[2] and faculty[3] in helping us achieve our goals for this project. On behalf of the Aqua Research Senior Design Group, thank you!

12 References

- Rodney Herrington, “2015 Requirements Document”,
- Tim Cushman, “K1 System Schematic”, Document: `K1K2SYS.SCH`
- Tim Cushman, “K1 K2 Disinfection Controller”, Document: `K1K2.SCH`
- Tim Cushman, “K1-K2 Controller Wiring Diagram”, Rev. 3/2020
- Tim Cushman, “Ox Tank Floats”, Document: `K2FLOAT.SCH`
- Tim Cushman, “Brine and Oxidant Generator Floats”, Document: `K1FLOAT.SCH`
- Tim Cushman, “K1 K2 Drivers”, Document: `K1K2DRV.SCH`
- Arduino Resources, Address: https://www.arduino.cc
- Adafruit I2C FRAM Resources, Address: https://www.adafruit.com/product/1895

[2] Rodney Herrington, Sponsor; Tim Cushman, Technical Mentor, Lois Warren, Technical Mentor

[3] Dr. Ramiro Jordan, Lead Instructor; Dr. Ganesh Balakrishnan, Assistant Instructor; Bradley Evans, Teaching Assistant

13 Source Code

13.1 final.ino

```
/*****************************************************************************************/
/*
/*  Authors:        Diego Chavez, David Kirby, John Quinlan
/*
/*  Description:    K1 disinfectant tank communication system
/*
/*  History:
/*                  v1.0 - 2015 Senior Design Group
/*                          Created LED displays for error detection
/*                  v2.0 - 2020 Senior Design Group
/*                          Added FRAM storage, LCD, and alert notification system
/*
/*****************************************************************************************/

//included library section
#include <Wire.h>
#include <LiquidCrystal_PCF8574.h>
#include <SoftwareSerial.h>
#include "I2C_string.h" //Local library for strings in FRAM
//#include <SIM800.h>

//Initialization of FRAMs as string reads
I2C_string fram = I2C_string();
I2C_string fram1 = I2C_string();

//LCD display with address 0x27
LiquidCrystal_PCF8574 lcd(0x27);

//For requirements 5 and 6
int ampsec = 0;
int liters = 0;

//Analog button values
const int LL_AB = 678;
const int LM_AB = 611;
const int RM_AB = 509;
const int RR_AB = 339;

//Analog inputs
int SCLA5 = 24;
int SDAA4 = 23;
int CV4 = 22;
int TEMP1 = 21;
int CIV = 20;
int A0_Mux = A0;
int ReadA0 = 0;
```

```
47
48  //Demuxing digital pins
49  const int D0A = 5;
50  const int D1B = 6;
51  const int D2C = 7;
52  const int D3D = 8;
53
54  //digital pins definiton
55  const int DBG0 = 9;
56  const int DBG1 = 10;
57  const int D0X0 = 11;
58  const int D0X1 = 12;
59  const int D0T0 = 13;
60  const int D0T1 = 14;
61
62  //Global language. Set default as English, can be changed through language select.
63  String Select = "Language Select";
64  String BeginDisinfection = "Begin Disinfection";
65  String ErrorMessage = "Error Message";
66  String QUIT = "Quit";
67
68  //Count the number of times the cell has been used.
69  //Will reverse polarity every 10 times in requirement 11.
70  uint8_t PolarityTest;
71
72  void setup(void)
73  {
74    //SMS set up, uncomment when fixed
75    //SendSMSsetup();
76    //Start FRAM i2c set up
77    //Baud rate
78    Serial.begin(9600);
79    while (!Serial)
80      ;
81    {
82      Serial.print("Waiting for Serial\n");
83    }
84    // Analog pin set up
85    pinMode(A0_Mux, INPUT);
86    pinMode(SCLA5, INPUT);
87    pinMode(SDAA4, INPUT);
88    pinMode(CV4, INPUT);
89    pinMode(TEMP1, INPUT);
90    pinMode(CIV, INPUT);
91
92    //Pin Outs and Ins ------------------------------------------------------------------
93    //The Mux pins
94    pinMode(D0A, OUTPUT);
95    pinMode(D1B, OUTPUT);
96    pinMode(D2C, OUTPUT);
97    pinMode(D3D, OUTPUT);
98
99    //Other digital pins
100   pinMode(DBG0, INPUT);
```

```
  pinMode(DBG1, INPUT);
  pinMode(DOX0, INPUT);
  pinMode(DOX1, INPUT);
  pinMode(DOT0, INPUT);
  pinMode(DOT1, INPUT);

  //Check the first FRAM
  if (fram1.begin(0x51))
  {
    Serial.println("Found I2C FRAM 0x51");
  }
  else
  {
    Serial.println("I2C FRAM not identified 0x51 ... check your connections?\r\n");
    alarm(0);
  }
  //Check the second FRAM
  if (fram.begin(0x50))
  { // you can stick the new i2c addr in here, e.g. begin(0x51);
    Serial.println("Found I2C FRAM 0x50");
  }
  else
  {
    Serial.println("I2C FRAM not identified 0x50... check your connections?\r\n");
    alarm(0);
  }
  //LCD set up ---------------------------------------------------------------------
  int error;
  Serial.println("LCD...\n");
  Serial.println("check for LCD");
  Wire.begin();
  Wire.beginTransmission(0x27);
  error = Wire.endTransmission();
  if (error == 0)
  {
    uint8_t test = fram.read8(0x0);
    Serial.println(": LCD found \n");
    lcd.begin(16, 2); // initialize the lcd
  }
  else
  {
    Serial.println(": LCD not found.");
    alarm(0);
  }
  lcd.setBacklight(255);
  lcd.home();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Hello LCD");
  lcd.setCursor(0, 1);
  delay(1000);
  lcd.clear();
  lcd.setBacklight(0);
}
```

```
155
156 void correctA0() //Corrects inconsistencies reading analog button
157 {
158   //int LL_AB=678;
159   //int LM_AB=611;
160   //int RM_AB=509;
161   //int RR_AB=339;
162   if (ReadA0 < 686 && ReadA0 > 669)
163   {
164     ReadA0 = LL_AB;
165   }
166   else if (ReadA0 < 616 && ReadA0 > 609)
167   {
168     ReadA0 = LM_AB;
169   }
170   else if (ReadA0 < 511 && ReadA0 > 504)
171   {
172     ReadA0 = RM_AB;
173   }
174   else if (ReadA0 < 346 && ReadA0 > 330)
175   {
176     ReadA0 = RR_AB;
177   }
178   else
179   {
180     ReadA0 = 0;
181   }
182 }
183 //Main loop ---------------------------------------------------------------------------
184 void loop(void)
185 {
186   ReadA0 = 0; //set the read A0 to zero.
187   delay(1000);
188   //default state selected
189   int SelectOption = 1;
190   LCDdis("->" + Select, BeginDisinfection, ErrorMessage, QUIT);
191
192   //wait until button pushed
193   while (ReadA0 != RM_AB)
194   {
195     ReadA0 = 0;
196     while (ReadA0 < 10)
197     {
198       updateMuxA0();
199       correctA0();
200       //Serial.println(ReadA0);
201       //delay(10);
202     }
203     //based on selected state go to function
204     if (ReadA0 == LL_AB)
205     {
206       SelectOption -= 1;
207     }
208     if (ReadA0 == LM_AB)
```

```
209       {
210         SelectOption += 1;
211       }
212       if (SelectOption < 0)
213       {
214         SelectOption = 4;
215       }
216       if (SelectOption > 4)
217       {
218         SelectOption = 1;
219       }
220
221       //Selects function based on state
222       switch (SelectOption)
223       {
224       case 1:
225         LCDdis("->" + Select, BeginDisinfection, ErrorMessage, QUIT);
226         break;
227       case 2:
228         LCDdis(Select, "->" + BeginDisinfection, ErrorMessage, QUIT);
229         break;
230       case 3:
231         LCDdis(Select, BeginDisinfection, "->" + ErrorMessage, QUIT);
232         break;
233       case 4:
234         LCDdis(Select, BeginDisinfection, ErrorMessage, "->" + QUIT);
235         break;
236       default:
237         //Should not be able to get here
238         Serial.println("error");
239         break;
240       }
241       //Serial.println(ReadA0);
242     }
243     //Goes to the selected function
244     switch (SelectOption)
245     {
246     case 1:
247       Hello_Phase();
248       break;
249     case 2:
250       Requirement10();
251       break;
252     case 3:
253       errorMSG();
254       break;
255     case 4:
256       //this is quit. needs a function to begin
257       LCDdis("Language Select", "Begin Disinfection", "Error Messages", "->Quit");
258       break;
259     default:
260       Serial.println("error");
261       break;
262     }
```

```
263 }
264
265 //Updates ReadA0 based on what user presses
266 void updateMuxA0()
267 {
268   ReadA0 = analogRead(A0_Mux);
269 }
270
271 //Takes four strings and displays them.
272 void LCDdis(String a, String b, String c, String d)
273 {
274   lcd.setBacklight(100);
275   lcd.home();
276   lcd.clear();
277   lcd.setCursor(0, 0);
278   lcd.print(a);
279   lcd.setCursor(0, 1);
280   lcd.print(b);
281   lcd.setCursor(0, 2);
282   lcd.print(c);
283   lcd.setCursor(0, 3);
284   lcd.print(d);
285   delay(100);
286 }
287
288 //Language select
289 void Hello_Phase(void)
290 {
291   ReadA0 = 0;
292   delay(1000);
293   int Selectt = 1;
294   LCDdis(Select, "", "<-   English   ->", "");
295
296   //While menu has not been selected
297   while (ReadA0 != RM_AB)
298   {
299     Read_AB();
300     correctA0();
301     if (ReadA0 == RR_AB)
302     {
303       return;
304     }
305
306     if (ReadA0 == LL_AB)
307     {
308       Selectt -= 1;
309     }
310     if (ReadA0 == LM_AB)
311     {
312       Selectt += 1;
313     }
314     if (Selectt < 0)
315     {
316       Selectt = 4;
```

```
317       }
318       if (Selectt > 4)
319       {
320         Selectt = 1;
321       }
322
323       //Displays language options
324       switch (Selectt)
325       {
326       case 1:
327         LCDdis(Select, "", "<-   English   ->", "");
328         break;
329       case 2:
330         LCDdis(Select, "", "<-    Spanish    ->", "");
331         break;
332       case 3:
333         LCDdis(Select, "", "<-    French    ->", "");
334         break;
335       case 4:
336         LCDdis(Select, "", "<-   Dutch    ->", "");
337         break;
338       default:
339         Serial.println("error");
340         break;
341       }
342     }
343     //Change the global strings based on language
344     switch (Selectt)
345     {
346     case 1:
347       //English
348       LanguageChange(0, 0);
349       break;
350     case 2:
351       //Spanish
352       LanguageChange(0, 80);
353       break;
354     case 3:
355       //French
356       LanguageChange(1, 0);
357       break;
358     case 4:
359       //Dutch
360       LanguageChange(1, 80);
361       break;
362     default:
363       Serial.println("error");
364       break;
365     }
366     return;
367   }
368
369   //Changes how to read the FRAM. Will need to change if you change languages in the FRAM
370   void LanguageChange(int a, int b)
```

```
{
  if (a == 0)
  {
    Select = fram.read_String(b + 0, 19);
    BeginDisinfection = fram.read_String(b + 20, 19);
    ErrorMessage = fram.read_String(b + 40, 19);
    QUIT = fram.read_String(b + 60, 19);
  }

  if (a == 1)
  {
    Select = fram1.read_String(b + 0, 19);
    BeginDisinfection = fram1.read_String(b + 20, 19);
    ErrorMessage = fram1.read_String(b + 40, 19);
    QUIT = fram1.read_String(b + 60, 19);
  }
}

//Waits for a button to be pressed by the user
void Read_AB(void)
{
  ReadA0 = 0;
  while (ReadA0 < 10)
  {
    updateMuxA0();
    //Serial.println(ReadA0);
  }
}

//Demuxing states. Will take argument int and output configuration in demux binary
void Actuators(int var)
{
  switch (var)
  {
  case 0: //Solenoid Valve 1
    digitalWrite(D0A, LOW);
    digitalWrite(D1B, LOW);
    digitalWrite(D2C, LOW);
    digitalWrite(D3D, LOW);
    break;

  case 1: //Solenoid Valve 2
    digitalWrite(D0A, HIGH);
    digitalWrite(D1B, LOW);
    digitalWrite(D2C, LOW);
    digitalWrite(D3D, LOW);
    break;

  case 2: //Solenoid Valve 3
    digitalWrite(D0A, LOW);
    digitalWrite(D1B, HIGH);
    digitalWrite(D2C, LOW);
    digitalWrite(D3D, LOW);
    break;
```

```
425
426   case 3: //Solenoid Valve 4
427     digitalWrite(D0A, HIGH);
428     digitalWrite(D1B, HIGH);
429     digitalWrite(D2C, LOW);
430     digitalWrite(D3D, LOW);
431     break;
432
433   case 4: //CELL ON
434     digitalWrite(D0A, LOW);
435     digitalWrite(D1B, LOW);
436     digitalWrite(D2C, HIGH);
437     digitalWrite(D3D, LOW);
438     break;
439
440   case 5: //Cell Polarity
441     digitalWrite(D0A, HIGH);
442     digitalWrite(D1B, LOW);
443     digitalWrite(D2C, HIGH);
444     digitalWrite(D3D, LOW);
445     break;
446
447   case 6: //Alarm
448     digitalWrite(D0A, LOW);
449     digitalWrite(D1B, HIGH);
450     digitalWrite(D2C, HIGH);
451     digitalWrite(D3D, LOW);
452     break;
453
454   case 7: //Green LED
455     digitalWrite(D0A, HIGH);
456     digitalWrite(D1B, HIGH);
457     digitalWrite(D2C, HIGH);
458     digitalWrite(D3D, LOW);
459     break;
460
461   case 8: //Yello LED
462     digitalWrite(D0A, LOW);
463     digitalWrite(D1B, LOW);
464     digitalWrite(D2C, LOW);
465     digitalWrite(D3D, HIGH);
466     break;
467
468   case 9: //red LED
469     digitalWrite(D0A, HIGH);
470     digitalWrite(D1B, LOW);
471     digitalWrite(D2C, LOW);
472     digitalWrite(D3D, HIGH);
473     break;
474
475   case 10: //empty do nothing
476     digitalWrite(D0A, LOW);
477     digitalWrite(D1B, HIGH);
478     digitalWrite(D2C, LOW);
```

```
479       digitalWrite(D3D, HIGH);
480       break;
481
482     case 11: //empty do nothing
483       digitalWrite(D0A, HIGH);
484       digitalWrite(D1B, HIGH);
485       digitalWrite(D2C, LOW);
486       digitalWrite(D3D, HIGH);
487       break;
488
489     case 15: //default off state
490       digitalWrite(D0A, HIGH);
491       digitalWrite(D1B, HIGH);
492       digitalWrite(D2C, LOW);
493       digitalWrite(D3D, HIGH);
494       break;
495
496     default:
497       Serial.print("Error. Incorrect demux state.\n");
498       delay(1);
499       break;
500     }
501   }
502
503   //Code based off the 2015 K1 requirements, but "interpreted" to fit the new board
504   //Requirement 1
505   int Requirement1(void)
506   {
507     LCDdis("Requirement 1", "started", "", "");
508     delay(300);
509
510     //Full state no error but alarm
511     if (readD(D0T0) == 1 && readD(D0T1) == 1)
512     {
513       alarm(1);
514       return 1;
515     }
516
517     //Error state defaults
518     if (readD(D0T0) == 0 && readD(D0T1) == 1 || readD(DBG0) == 0 &&
519         readD(DBG1) == 1 || readD(D0X0) == 0 && readD(D0X1) == 1)
520     {
521       alarm(3);
522       return 1;
523     }
524     //Empty state = tell user to check tanks
525     if (readD(DBG0) == 0 && readD(DBG1) == 0 || readD(D0X0) == 0 && readD(D0X1) == 0)
526     {
527       alarm(2);
528       return 1;
529     }
530
531     //Start state = no error
532     if (readD(D0T1) == 0 && readD(D0T0) == 1 || readD(D0T1) == 0 && readD(D0T0) == 0)
```

```
533   {
534     Serial.print("start state in req1\n"); //Turn system on
535     delay(5000);
536     Actuators(4); //Turn on cell
537     return 0;     //Return with no error
538   }
539 }
540
541 int Requirement2(void)
542 {
543   LCDdis("Requirement 2", "started", "", "");
544   Actuators(0);             //Turn on solenoid valve 1
545   while (readD(DBG1) == 1) //while empty wait to fill
546   {
547     //do nothing}
548     delay(160);
549     Serial.print("in Req2 waiting for BG1\n");
550   }
551   //error state
552   if (readD(DBG1) == 1 && readD(DBG0) == 0)
553   {
554     alarm(3);
555     return 1;
556   }
557   Serial.print("Req2 success\n");
558   //no error reported. turn off everything
559   Actuators(15);
560   return 0;
561 }
562
563 int Requirement3(void)
564 {
565   LCDdis("Requirement 3", "started", "", "");
566   delay(1600);
567   Actuators(1);                                  //Turn on solenoid valve 1
568   while (readD(D0X0) != 1 && readD(D0X1) != 1) //While not full
569   {
570
571     if (readD(D0X0) == 0 && readD(D0X1) == 1) //error state in sensors
572     {
573       alarm(3);
574       return 1;
575     }
576     delay(1);
577   }
578   Actuators(15);
579   return 0;
580 }
581
582 void Requirement4(void)
583 {
584   LCDdis("Requirement 4", "started", "", "");
585   //Cell on
586   Actuators(4);
```

```
587   delay(5000);
588   //Actuators(15);
589 }
590
591 //Read amps-sec and output as liters. Not defined yet by Tim
592 int Requirement56(void)
593 {
594   LCDdis("Requirement 56", "started", "", "");
595   int Temp = 0;
596   int sec = 0;
597
598   //While amp-sec is less than a certain value
599   while (ampsec < 100)
600   {
601     delay(1000);
602     Temp = analogRead(CIV);
603     //Error if CIV amperage is very low
604     if (analogRead(Temp <= 1))
605     {
606       alarm(4);
607       return 1;
608     }
609     //Error if CIV is way to high
610     if (analogRead(Temp >= 5))
611     {
612       alarm(5);
613       return 1;
614     }
615
616     ampsec = Temp * sec;
617     //100? 100 is supposed to be charge
618     sec += 1;
619   }
620   return 0;
621 }
622
623 //Turn off everything
624 void Requirement7(void)
625 {
626   LCDdis("Requirement 10", "started", "", "");
627   Actuators(15);
628 }
629
630 int Requirement89(void)
631 {
632   LCDdis("Requirement 89", "started", "", "");
633   Actuators(2);                                  //Turn on solenoid valve 3
634   while (readD(D0X0) != 0 && readD(D0X1) != 0) //While not empty
635   {
636     //Checking errors in sensors
637     if (readD(D0T0) == 0 && readD(D0T1) == 1 || readD(DBG0) == 0 &&
638         readD(DBG1) == 1 || readD(D0X0) == 0 && readD(D0X1) == 1)
639     {
640       alarm(3);
```

```
641         return 1;
642       }
643       delay(1000);
644     }
645     //Turn off everything and report no error
646     Actuators(15);
647     return 0;
648   }
649   //Change polarity every ten uses. Saves to fram and counts between shut-offs
650   void Requirement11(void)
651   {
652     LCDdis("Requirement 11", "started", "", "");
653     //Read from address in FRAM
654     PolarityTest = fram.read8(0xFF);
655     if (PolarityTest > 9) //If greater then 9, reset
656     {
657       Actuators(10);
658       PolarityTest = 0;
659       delay(1000);
660       Actuators(15);
661     }
662     PolarityTest += 1;
663     fram.write8(0xFF, PolarityTest); //Save state to FRAM
664   }
665   //Generating
666   void Requirement10(void)
667   {
668     LCDdis("Requirement 10", "started", "", "");
669     delay(3000);
670     //Tank full
671     if (readD(DOT1) == 1 && readD(DOT0) == 1)
672     {
673       LCDdis("Oxidant Tank Full", "Empty Please", "", "");
674       return;
675     }
676     //Error states for sensors
677     if (readD(DOT0) == 0 && readD(DOT1) == 1 || readD(DBG0) == 0 &&
678         readD(DBG1) == 1 || readD(DOX0) == 0 && readD(DOX1) == 1)
679     {
680       alarm(3);
681       return;
682     }
683     //Error states for empty tanks
684     if (readD(DBG0) == 0 && readD(DBG1) == 0 || readD(DOX0) == 0 && readD(DOX1) == 0)
685     {
686       alarm(2);
687       return;
688     }
689     //While OT tank is not full, generate
690     while (readD(DOT1) != 1 && readD(DOT0) != 1)
691     {
692       //If a state returns 1, it will exit this function
693       if (Requirement1() == 1)
694       {
```

```
695         return;
696       }
697       if (Requirement2() == 1)
698       {
699         return;
700       }
701       if (Requirement3() == 1)
702       {
703         return;
704       }
705       Requirement4();
706       if (Requirement56() == 1)
707       {
708         return;
709       }
710       Requirement7();
711       if (Requirement89() == 1)
712       {
713         return;
714       }
715       Requirement11();
716     }
717   }
718   //Reads a digital pin
719   int readD(int digital)
720   {
721     return digitalRead(digital);
722   }
723
724   //Converts the ampseconds to liters for output
725   void ampsecliters(void)
726   {
727     if (ampsec > 2)
728     {
729       liters = 20;
730     }
731     if (ampsec > 1.5)
732     {
733       liters = 15;
734     }
735     if (ampsec > 1)
736     {
737       liters = 10;
738     }
739     if (ampsec > .5)
740     {
741       liters = 5;
742     }
743   }
744   //Activate alarm if conditions are met
745   void alarm(int a)
746   {
747     uint8_t temp = 0;
748     //Activate alarm
```

```
749   Actuators(6);
750   //Turn on red light
751   Actuators(10);
752   //Record error, tell user what is wrong, send SMS
753   switch (a)
754   {
755   //Start up error
756   case 0:
757     LCDdis0();
758     temp = fram1.read8(0xF8);
759     temp += 1;
760     fram1.write8(0xF8, temp);
761     //sendSMS((char) 0xF8);
762     break;
763
764   //Full alarm
765   case 1:
766     LCDdis1();
767     temp = fram1.read8(0xF9);
768     temp += 1;
769     fram1.write8(0xF9, temp);
770     //sendSMS((char) 0xF9);
771     break;
772
773   //Empty error
774   case 2:
775     LCDdis2();
776     temp = fram1.read8(0xFA);
777     temp += 1;
778     fram1.write8(0xFA, temp);
779     //sendSMS((char) 0xFA);
780     break;
781
782   //Sensor error illegal state
783   case 3:
784     LCDdis3();
785     temp = fram1.read8(0xFB);
786     temp += 1;
787     fram1.write8(0xFB, temp);
788     //sendSMS((char) 0xFB);
789     break;
790
791   //Power supply close-to-zero error
792   case 4:
793     LCDdis4();
794     temp = fram1.read8(0xFC);
795     temp += 1;
796     fram1.write8(0xFC, temp);
797     //sendSMS((char) 0xFC);
798     break;
799
800   //Power supply over amperage
801   case 5:
802     LCDdis5();
```

```
803       temp = fram.read8(0xFD);
804       temp += 1;
805       fram1.write8(0xFD, temp);
806       //sendSMS((char) 0xFD);
807       break;
808
809     //Clear state for whatever reason
810     default:
811       fram1.write8(0xF8, 0);
812       fram1.write8(0xF9, 0);
813       fram1.write8(0xFA, 0);
814       fram1.write8(0xFB, 0);
815       fram1.write8(0xFC, 0);
816       fram1.write8(0xFD, 0);
817       break;
818       //sendSMS((char) a);
819     }
820   }
821   //User presses this from main menu.
822   //Tells user the recorded errors, what alarms went off, and what to do.
823   void errorMSG(void)
824   {
825     //Default state
826     ReadA0 = 0;
827     delay(1000);
828     int Selectt = 1;
829     LCDdis(ErrorMessage, "<-Alarm 0->", "Start up error", "number " +
830           (String)fram1.read8(0xF8));
831     //While main menu is not pressed
832     while (ReadA0 != RM_AB)
833     {
834       Read_AB();
835       correctA0();
836       if (ReadA0 == RR_AB)
837       {
838         return;
839       }
840       if (ReadA0 == LL_AB)
841       {
842         Selectt -= 1;
843       }
844       if (ReadA0 == LM_AB)
845       {
846         Selectt += 1;
847       }
848       if (Selectt < 0)
849       {
850         Selectt = 8;
851       }
852       if (Selectt > 8)
853       {
854         Selectt = 1;
855       }
856
```

```
857       switch (Selectt)
858       {
859       case 1:
860         LCDdis(ErrorMessage, "<-Alarm 0->", "Start up error", "number " +
861               (String)fram1.read8(0xF8));
862         break;
863       case 2:
864         LCDdis(ErrorMessage, "<-Alarm 1->", "Full Error", "number " +
865               (String)fram1.read8(0xF9));
866         break;
867       case 3:
868         LCDdis(ErrorMessage, "<-Alarm 2->", "Tank Empty", "number " +
869               (String)fram1.read8(0xFA));
870         break;
871       case 4:
872         LCDdis(ErrorMessage, "<-Alarm 3->", "Sensor Error", "number " +
873               (String)fram1.read8(0xFB));
874         break;
875       case 5:
876         LCDdis(ErrorMessage, "<-Alarm 4->", "Power Supply Low", "number " +
877               (String)fram1.read8(0xFC));
878         break;
879       case 6:
880         LCDdis(ErrorMessage, "<-Alarm 5->", "Power Supply High", "number " +
881               (String)fram1.read8(0xFD));
882         break;
883       case 7:
884         LCDdis(ErrorMessage, "<-       ->", "Send Errors", "To Aqua Research");
885         break;
886       case 8:
887         LCDdis(ErrorMessage, "<-       ->", "Clear all Errors", "number " +
888               (String)(fram1.read8(0xF8) + fram1.read8(0xF9) + fram1.read8(0xFA) +
889               fram1.read8(0xFB) + fram1.read8(0xFC) + fram1.read8(0xFD)));
890         break;
891       default:
892         Serial.println("error");
893         break;
894       }
895     }
896     switch (Selectt)
897     {
898     case 1:
899       //Alarm 0 message
900       LCDdis0();
901       break;
902     case 2:
903       //Alarm 1 message
904       LCDdis1();
905       break;
906     case 3:
907       //Alarm 2 message
908       LCDdis2();
909       break;
910     case 4:
```

```
911       //Alarm 3 message
912       LCDdis3();
913       ReadA0 = 0;
914       while (ReadA0 < 10)
915       {
916         Read_AB();
917       }
918       break;
919     case 5:
920       //Alarm 4 message
921       LCDdis4();
922       break;
923     case 6:
924       //Alarm 5 message
925       LCDdis5();
926       break;
927     case 7:
928       //sends alle rrrors to aqua research
929       for (int i = 0xF8; i < 0xFE; i++)
930       {
931         if (fram1.read8(i) != 0)
932         {
933           delay(2);
934           //sendSMS((char)i)
935         }
936       }
937       delay(1);
938       break;
939     case 8:
940       //clear all errors
941       alarm(15);
942       break;
943     default:
944       Serial.println("error");
945       break;
946     }
947     return;
948   }
949
950   //Alarm 0 message.
951   void LCDdis0()
952   {
953     lcd.setBacklight(100);
954     lcd.home();
955     lcd.clear();
956     lcd.setCursor(0, 0);
957     lcd.print("Start up Error");
958     lcd.setCursor(0, 1);
959     lcd.print("");
960     lcd.setCursor(0, 2);
961     lcd.print("");
962     lcd.setCursor(0, 3);
963     lcd.print("");
964     delay(100);
```

```
965   ReadA0 = 0;
966   while (ReadA0 < 10)
967   {
968     Read_AB();
969   }
970 }
971 //Alarm 1 message
972 void LCDdis1()
973 {
974   lcd.setBacklight(100);
975   lcd.home();
976   lcd.clear();
977   lcd.setCursor(0, 0);
978   lcd.print("Empty Full Tank");
979   lcd.setCursor(0, 1);
980   lcd.print("Check Tank:");
981   lcd.setCursor(0, 2);
982   lcd.print("Brine, Oxidant, Or");
983   lcd.setCursor(0, 3);
984   lcd.print("Electrolyte Tanks");
985   delay(100);
986   ReadA0 = 0;
987   while (ReadA0 < 10)
988   {
989     Read_AB();
990   }
991 }
992 //Alarm 2 message
993 void LCDdis2()
994 {
995   lcd.setBacklight(100);
996   lcd.home();
997   lcd.clear();
998   lcd.setCursor(0, 0);
999   lcd.print("Check for Empty Tank");
1000  lcd.setCursor(0, 1);
1001  lcd.print("Check Tank");
1002  lcd.setCursor(0, 2);
1003  lcd.print("Brine and");
1004  lcd.setCursor(0, 3);
1005  lcd.print("Electrolyte");
1006  delay(100);
1007  ReadA0 = 0;
1008  while (ReadA0 < 10)
1009  {
1010    Read_AB();
1011  }
1012 }
1013 //Alarm 3 message
1014 void LCDdis3()
1015 {
1016   lcd.setBacklight(100);
1017   lcd.home();
1018   lcd.clear();
```

```
1019    lcd.setCursor(0, 0);
1020    lcd.print("Senser Error");
1021    lcd.setCursor(0, 1);
1022    lcd.print("Check all Tank");
1023    lcd.setCursor(0, 2);
1024    lcd.print("Sensors");
1025    lcd.setCursor(0, 3);
1026    lcd.print("");
1027    delay(100);
1028    ReadA0 = 0;
1029    while (ReadA0 < 10)
1030    {
1031      Read_AB();
1032    }
1033  }
1034  //Alarm 4 message
1035  void LCDdis4()
1036  {
1037    lcd.setBacklight(100);
1038    lcd.home();
1039    lcd.clear();
1040    lcd.setCursor(0, 0);
1041    lcd.print("No Current");
1042    lcd.setCursor(0, 1);
1043    lcd.print("From Power Supply");
1044    lcd.setCursor(0, 2);
1045    lcd.print("");
1046    lcd.setCursor(0, 3);
1047    lcd.print("");
1048    delay(100);
1049    ReadA0 = 0;
1050    while (ReadA0 < 10)
1051    {
1052      Read_AB();
1053    }
1054  }
1055  //Alarm 5 message
1056  void LCDdis5()
1057  {
1058    lcd.setBacklight(100);
1059    lcd.home();
1060    lcd.clear();
1061    lcd.setCursor(0, 0);
1062    lcd.print("Excess Current");
1063    lcd.setCursor(0, 1);
1064    lcd.print("Power Supply Too");
1065    lcd.setCursor(0, 2);
1066    lcd.print("Much Amperage");
1067    lcd.setCursor(0, 3);
1068    lcd.print("");
1069    delay(100);
1070    ReadA0 = 0;
1071    while (ReadA0 < 10)
1072    {
```

```
    Read_AB();
  }
}
/* When sim card is hooked up properly uncomment
//Set up for sms
void SendSMSsetup(void)
{
  SIM.begin(9600);
  delay(100);
  SIM.pinCode(GET);
  // WARNING! Be certain that you input the correct pin code!
  if (SIM.reply("SIM PIN")) SIM.pinCode(SET, code);
}
//Sends the SMS
void sendSMS(char error){
  SIM.smsFormat(SET, "1");
  SIM.smsSend(addr, error);
}
*/
```

13.2 I2C_string.cpp

```
/**************************************************************************/
/*!
    @file     Adafruit_FRAM_I2C.cpp
    @author   KTOWN (Adafruit Industries)
    @license  BSD (see license.txt)

    Driver for the Adafruit I2C FRAM breakout.

    Adafruit invests time and resources providing this open source code,
    please support Adafruit and open-source hardware by purchasing
    products from Adafruit!

    @section  HISTORY

    v1.0 - First release
    v1.1 - 16-bit expansion using bitwise operation
*/
/**************************************************************************/
//#include <avr/pgmspace.h>
//#include <util/delay.h>
#include <stdlib.h>
#include <math.h>

#include "I2C_string.h"

/*=========================================================================*/
/*                            CONSTRUCTORS                                 */
/*=========================================================================*/

/**************************************************************************/
/*!
    Constructor
*/
/**************************************************************************/
I2C_string::I2C_string(void)
{
  _framInitialised = false;
}

/*=========================================================================*/
/*                           PUBLIC FUNCTIONS                              */
/*=========================================================================*/

/**************************************************************************/
/*!
    Initializes I2C and configures the chip (call this function before
    doing anything else)
*/
/**************************************************************************/
boolean I2C_string::begin(uint8_t addr)
```

```
51  {
52
53    i2c_addr = addr;
54    Wire.begin();
55
56    /* Make sure we're actually connected */
57    uint16_t manufID, prodID;
58    getDeviceID(&manufID, &prodID);
59    if (manufID != 0x00A)
60    {
61      Serial.print("Unexpected Manufacturer ID: 0x");
62      Serial.println(manufID, HEX);
63      return false;
64    }
65    if (prodID != 0x510)
66    {
67      Serial.print("Unexpected Product ID: 0x");
68      Serial.println(prodID, HEX);
69      return false;
70    }
71
72    /* Everything seems to be properly initialised and connected */
73    _framInitialised = true;
74
75    return true;
76  }
77
78  /**************************************************************************/
79  /*!
80      @brief  Writes a byte at the specific FRAM address
81
82      @params[in] i2cAddr
83                  The I2C address of the FRAM memory chip (1010+A2+A1+A0)
84      @params[in] framAddr
85                  The 16-bit address to write to in FRAM memory
86      @params[in] i2cAddr
87                  The 8-bit value to write at framAddr
88  */
89  /**************************************************************************/
90  void I2C_string::write8 (uint16_t framAddr, uint8_t value)
91  {
92    Wire.beginTransmission(i2c_addr);
93    Wire.write(framAddr >> 8);
94    Wire.write(framAddr & 0xFF);
95    Wire.write(value);
96    Wire.endTransmission();
97  }
98
99  /**************************************************************************/
100 /*!
101     @brief  Reads an 8-bit value from the specified FRAM address
102
103     @params[in] i2cAddr
104                 The I2C address of the FRAM memory chip (1010+A2+A1+A0)
```

```
105     @params[in] framAddr
106                 The 16-bit address to read from in FRAM memory
107
108     @returns    The 8-bit value retrieved at framAddr
109 */
110 /**************************************************************************/
111 uint8_t I2C_string::read8 (uint16_t framAddr)
112 {
113   Wire.beginTransmission(i2c_addr);
114   Wire.write(framAddr >> 8);
115   Wire.write(framAddr & 0xFF);
116   Wire.endTransmission();
117
118   Wire.requestFrom(i2c_addr, (uint8_t)1);
119
120   return Wire.read();
121 }
122
123 /**************************************************************************/
124 /*!
125     @brief  Reads the Manufacturer ID and the Product ID frm the IC
126
127     @params[out]  manufacturerID
128                   The 12-bit manufacturer ID (Fujitsu = 0x00A)
129     @params[out]  productID
130                   The memory density (bytes 11..8) and proprietary
131                   Product ID fields (bytes 7..0). Should be 0x510 for
132                   the MB85RC256V.
133 */
134 /**************************************************************************/
135 void I2C_string::getDeviceID(uint16_t *manufacturerID, uint16_t *productID)
136 {
137   uint8_t a[3] = { 0, 0, 0 };
138   uint8_t results;
139
140   Wire.beginTransmission(MB85RC_SLAVE_ID >> 1);
141   Wire.write(i2c_addr << 1);
142   results = Wire.endTransmission(false);
143
144   Wire.requestFrom(MB85RC_SLAVE_ID >> 1, 3);
145   a[0] = Wire.read();
146   a[1] = Wire.read();
147   a[2] = Wire.read();
148
149   /* Shift values to separate manuf and prod IDs */
150   /* See p.10 of http://www.fujitsu.com/downloads/MICRO/fsa/pdf/products/memory/fram/MB85RC256V-DS501-0
151   *manufacturerID = (a[0] << 4) + (a[1]  >> 4);
152   *productID = ((a[1] & 0x0F) << 8) + a[2];
153 }
154
155 /**************************************************************************/
156 /*!
157     @brief  Writes a 16-bit value to the specified FRAM address
158
```

```
159     @params[in] framAddr
160                 The I2C address of the FRAM memory chip (1010+A2+A1+A0)
161     @params[in] value1
162                 The 16-bit value to be written
163 
164 */
165 /**************************************************************************/
166 void I2C_string::write16(uint16_t framAddr, uint16_t value1)
167 {
168         int c_address = framAddr+framAddr;
169         uint16_t low,high;
170         low=value1 & 0x00FF;
171         high=value1 & 0xFF00;
172         high= high >> 8;
173 
174         write8(c_address, low);
175         write8(c_address+1, high);
176 }
177 
178 /**************************************************************************/
179 /*!
180     @brief  Reads a 16-bit value from the specified FRAM address
181 
182     @params[in] framAddr
183                 The 8-bit address to read from in FRAM memory
184 
185     @returns    The 16-bit value retrieved at framAddr
186 */
187 /**************************************************************************/
188 uint16_t I2C_string::read16(uint16_t framAddr)
189 {
190         int c_address = framAddr+framAddr;
191         uint16_t low;
192         uint16_t high;
193         low = read8(c_address);//& 0xff;
194         high = read8(c_address+1);// << 8;
195 
196         uint16_t temp1 = 256*high+low;
197 
198         return twos_comp_check(temp1);
199 }
200 
201 /**************************************************************************/
202 /*!
203     @brief  Checks to see if signed value is negative or positive
204 
205     @params[in] number
206                 The value to be checked
207 
208     @returns    The value as either a correct negative number or a positive
209 */
210 /**************************************************************************/
211 uint16_t I2C_string::twos_comp_check(uint16_t number)
212 {
```

```
213         uint16_t numcheck = 0, number2 = number&0xFFFF;
214
215         numcheck = number2 >>15;
216
217
218         if(numcheck == 0)
219         {
220                 return number;
221         }
222         else
223         {
224                 number2 = number2^0xFFFF;
225                 return -(number2+1);
226         }
227
228 }
229
230 /*************************************************************************/
231 /*!
232     @brief  Reads a string from the specified FRAM address
233
234     @params[in] Addr
235                 The I2C address of the FRAM memory chip (1010+A2+A1+A0)
236     @params[in] length
237                 The length of the string being read
238
239     @returns    The string read from starting address ending at length
240 */
241 /*************************************************************************/
242 String I2C_string::read_String(int Addr, int length)
243 {
244         char temparray[length];
245         for(int i=0;i<length;i++)
246         {
247                 temparray[i]= (char)read8(Addr+i);
248         }
249         String stemp(temparray);
250
251         return stemp;
252
253 }
254
255 /*************************************************************************/
256 /*!
257     @brief  Writes string to the specified FRAM address
258
259     @params[in] Addr
260                 The I2C address of the FRAM memory chip (1010+A2+A1+A0)
261     @params[in] input
262                 The String to be written to FRAM memory
263
264 */
265 /*************************************************************************/
266 void I2C_string::write_String(int Addr, String input)
```

```
{
        int numBytes;
        char cbuff[input.length()+1];
        input.toCharArray(cbuff,input.length()+1);
        for (int i = 0; i < input.length()+1; i++) {
                write8(Addr + i,cbuff[i]);
        }

}
```

13.3 I2C_string.h

```
/**************************************************************************/
/*!
    @file     Adafruit_FRAM_I2C.h
    @author   KTOWN (Adafruit Industries)

    @section LICENSE

    Software License Agreement (BSD License)

    Copyright (c) 2013, Adafruit Industries
    All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, are permitted provided that the following conditions are met:
    1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
    2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.
    3. Neither the name of the copyright holders nor the
    names of its contributors may be used to endorse or promote products
    derived from this software without specific prior written permission.

    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ''AS IS'' AND ANY
    EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY
    DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
    ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

    Revision
    1. 16-bit expansion using bitwise operation
*/
/**************************************************************************/
#ifndef _I2C_string_H_
#define _I2C_string_H_

#if ARDUINO >= 100
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif

#include <Wire.h>

#define MB85RC_DEFAULT_ADDRESS        (0x50) /* 1010 + A2 + A1 + A0 = 0x50 default */
```

```
51 #define MB85RC_SLAVE_ID       (0xF8)
52
53 class I2C_string {
54  public:
55   I2C_string(void);
56
57   boolean  begin(uint8_t addr = MB85RC_DEFAULT_ADDRESS);
58   void     write8 (uint16_t framAddr, uint8_t value);
59   uint8_t  read8  (uint16_t framAddr);
60   void     getDeviceID(uint16_t *manufacturerID, uint16_t *productID);
61
62   //Added functions
63         void write16(uint16_t framAddr, uint16_t);
64         uint16_t read16(uint16_t framAddr);
65         uint16_t twos_comp_check(uint16_t);
66
67         String read_String(int, int);
68         void write_String(int, String);
69
70  private:
71   uint8_t i2c_addr;
72   boolean _framInitialised;
73 };
74
75 #endif
```